Certified Decision Procedures for Two-Counter Machines

Andrej Dudenhefner
TU Dortmund, Germany

Abstract

Two-counter machines, pioneered by Minsky in the 1960s, constitute a particularly simple, universal model of computation. Universality of *reversible* two-counter machines (having a right-unique step relation) has been shown by Morita in the 1990s. Therefore, the halting problem for reversible two-counter machines is undecidable. Surprisingly, this statement is specific to certain instruction sets of the underlying machine model.

In the present work we consider two-counter machines (CM2) with instructions $\mathtt{inc}_c$ (increment counter c, go to next instruction), $\mathtt{dec}_c\, q$ (if counter c is zero, then go to next instruction, otherwise decrement counter c and go to instruction q). While the halting problem for CM2 is undecidable, we give a decision procedure for the halting problem for reversible CM2, contrasting Morita's result.

We supplement our result with decision procedures for uniform boundedness (is there a uniform bound on the number of reachable configurations?) and uniform mortality (is there a uniform bound on the number of steps in any run?) for CM2.

Termination and correctness of each presented decision procedure is certified using the Coq proof assistant. In fact, both the implementation and certification is carried out simultaneously using the tactic language of the Coq proof assistant. Building upon existing infrastructure, the mechanized decision procedures are contributed to the Coq library of undecidability proofs.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases constructive mathematics, computability theory, decidability, counter automata, mechanization, Coq

Digital Object Identifier 10.4230/LIPIcs.FSCD.2022.16

Supplementary Material *Software (Source Code)*: `https://github.com/uds-psl/coq-library-undecidability/tree/coq-8.15/theories/CounterMachines`

1 Introduction

In the early 1960s, Minsky has shown the universality of two-tape, read-only Turing machines [18]. As a result, two-counter machines (originally called "program-machines" [19, Table 11.1-1]) emerged as a particularly simple, universal model of computation. A two-counter machine stores data in two registers, each containing a natural number. While the particular instruction sets may vary (for an overview see [13, Section 2]), common machine instructions are: counter increment, counter decrement (possibly including a conditional jump), and counter zero test (including a conditional jump). Due to the arithmetically simple nature of machine instructions, two-counter machines are easily simulated by other machine models. This often leads to small, universal constructions [11]. Another prominent example, which relies on two-counter machines, is the nested simulation technique, invented by Hooper for Turing machine immortality [10, 12]. Besides the halting problem for two-counter machines, mortality and boundedness problems [14] constitute useful tools in the area of model checking.

In the research field of *reversible computing*, which considers "backward deterministic" computation, universality of reversible two-counter machines was shown by Morita [20]. By reversible simulation, this milestone result has immediate implications for other models of computation (for an overview see [21, 22]), and group theory [24].

Commonly, the instruction set of the underlying two-counter machine model is chosen suitably for the individual results. Key results such as (reversible) universality are transferred tacitly between instruction sets. Surprisingly, this is not always possible. The most prominent example is the decidability of the halting problem for "program-machines" with two registers (Remark 1), as originally given by Minsky.

▶ **Remark 1.** Consider the exact definition of "program-machines" given by Minsky [19, Table 11.1-1]. That is, lists of instructions from the following instruction set:

- set counter c to zero, go to next instruction
- increment counter c, go to next instruction
- if counter c is positive, then decrement counter c and go to next instruction, otherwise go to instruction n
- halt

The halting problem for the above machine model with two counters is decidable[1]. Conditional control flow in the above machine model is based on failing counter decrement instructions. That is, if the value of one of the counters is zero. For example, given a program of length n, if both counters are larger than n, then (unable to go to any previously visited instruction) the program halts after at most n steps. Therefore, for any run, values of at least one of the counters are drawn from a finite set. Overall, the above machine model with two counters is not universal.

In his argument, Minsky necessarily includes an additional, unconditional jump instruction [19, Chapter 14] in order to have a universal machine model with two counters.

In the present work we consider two-counter machines (CM2) as list of instructions from the instruction set: $\mathtt{inc}_c$ (increment counter c, go to next instruction), $\mathtt{dec}_c\, q$ (if counter c is zero, then go to next instruction, otherwise decrement counter c and go to instruction q). CM2, relying on its arguably minimal instruction set, plays a key role in mechanized undecidability results [8] (such as Hilbert's tenth problem [15] and semi-unification [2]). The key difference to Minsky's "program-machines" is that the conditional jump is on successful (instead of failed) counter decrement. In contrast to Remark 1, this instruction set suffices for an undecidable halting problem (Theorem 6). However, this instruction set does *not* suffice for an undecidable reversible halting problem (Theorem 21), which is our main result. Intuitively, conditional control flow for the above instruction set is too restricted in the reversible setting, and does not allow for nested loops. As a consequence, control flow for reversible CM2 can be modeled by a finite state automaton, resulting in a decision procedure for termination.

In addition, we consider boundedness and mortality problems for CM2. First, we contrast undecidability of total boundedness [14] (is for any configuration the number of reachable configurations finite?) for CM2 with a decision procedure for uniform boundedness (is there a uniform bound on the number of reachable configurations?). Second, we contrast undecidability of total mortality [10] for CM2 (does every run eventually halt?) with a decision procedure for uniform mortality (is there a uniform bound on the number of steps in any run?). While decidability of uniform mortality is known [12, Theorem 2], the more complex decidability of uniform boundedness is hitherto only hinted at [1, Remark 28]. Additionally, the decision algorithms provided in the present work use explicit upper bounds, and are well-suited for complexity-theoretic analysis.

[1] The certified decision procedure is mechanized as the computable Boolean function `decide : Mpm2 * Config -> bool` in `theories/MinskyMachines/MPM2_HALT_dec.v` in the Coq library of undecidability proofs [8].

Formal specification, termination certification, and verification of correctness of the presented decision procedures is carried out using the Coq proof assistant. Following the compelling argument by Forster [3], the Coq proof assistant is excellently positioned to argue about computability-theoretic properties of decision problems. For a predicate P over a domain X, a *decision procedure* is a Boolean function $f : X \to \mathbb{B}$ such that for all $x \in X$ we have $(f(x) = \texttt{true}) \Leftrightarrow P(x)$. Any axiom-free implementation of a decision procedure f in Coq entails a termination argument for f on any input. As added benefit, the correctness proof of f can be given and verified mechanically in Coq as part of the definition of f. Effective implementations of the individual decision procedures can be obtained using the `Extraction` framework [17]. Overall, the approach taken in the present work is well positioned in the intersection of computability theory and constructive mathematics.

The growing Coq library of undecidability proofs [8] already contains a plethora of negative and positive[2] computability results. Since CM2 is a prominent decision problem in the library, we build upon the existing infrastructure to contribute the decision procedures in the present work to the library.

Organization. The remainder of the present work is organized as follows.

Section 2: Preliminary definitions and properties of two-counter machines (CM2).
Section 3: Decision procedure for the halting problem for reversible CM2 (Theorem 21).
Section 4: Decision procedure for the uniform boundedness problem for CM2 (Theorem 40).
Section 5: Decision procedure for the uniform mortality problem for CM2 (Theorem 49).
Section 6: Remarks on the mechanization in the Coq proof assistant of the above decision procedures, and the contribution to the Coq library of undecidability proofs.
Section 7: Concluding remarks.

2 Two-Counter Machine Preliminaries

In this section we recollect the definition of two-counter machines (CM2, Definition 2) and the undecidability of the corresponding halting problem (Theorem 6). The main benefit of CM2 is its small, universal instruction set (cf. [13, Section 2]). As a result, CM2 allows for compact proofs, and plays a key role in mechanized undecidability results [8].

▶ **Definition 2** (Two-Counter Machine (CM2)). A *two-counter machine* $\mathcal{M}$ is a list of *instructions* of shape either $\texttt{inc}_0$, $\texttt{inc}_1$, $\texttt{dec}_0\, q$, or $\texttt{dec}_1\, q$, where $q \in \mathbb{N}$ is a *program index*.

A *configuration* of $\mathcal{M}$ is of shape $(p, (a, b))$, where $p \in \mathbb{N}$ is the current *program index* and $a, b \in \mathbb{N}$ are the current *counter values*.

The *step relation* of $\mathcal{M}$ on configurations, written $(\longrightarrow_{\mathcal{M}})$, is given by

- if $\texttt{inc}_0$ is the p-th instruction of $\mathcal{M}$, then $(p, (a, b)) \longrightarrow_{\mathcal{M}} (p + 1, (a + 1, b))$
- if $\texttt{inc}_1$ is the p-th instruction of $\mathcal{M}$, then $(p, (a, b)) \longrightarrow_{\mathcal{M}} (p + 1, (a, b + 1))$
- if $\texttt{dec}_0\, q$ is the p-th instruction of $\mathcal{M}$, then $(p, (0, b)) \longrightarrow_{\mathcal{M}} (p + 1, (0, b))$ and $(p, (a + 1, b)) \longrightarrow_{\mathcal{M}} (q, (a, b))$
- if $\texttt{dec}_1\, q$ is the p-th instruction of $\mathcal{M}$, then $(p, (a, 0)) \longrightarrow_{\mathcal{M}} (p + 1, (a, 0))$ and $(p, (a, b + 1)) \longrightarrow_{\mathcal{M}} (q, (a, b))$
- otherwise, we say that $(p, (a, b))$ *halts*

[2] For example, Spies and Forster [26] contrast undecidability of higher-order unification with a decision procedure for first-order unification in Coq.

The *reachability relation* of $\mathcal{M}$ on configurations, written $(\longrightarrow_{\mathcal{M}}^{*})$, is the reflexive, transitive closure of $(\longrightarrow_{\mathcal{M}})$. The transitive closure of $(\longrightarrow_{\mathcal{M}})$ is denoted by $(\longrightarrow_{\mathcal{M}}^{+})$.

A configuration x is *terminating* in $\mathcal{M}$, if we have $x \longrightarrow_{\mathcal{M}}^{*} y$ for some halting configuration y.

The length of $\mathcal{M}$ is denoted by $|\mathcal{M}|$.

Comparing the above Definition 2 to Minsky's original definition [19, Table 11.1-1], the main difference is that the conditional jump is performed on a successful (instead of failed) counter decrement. In the setting with only two counters this difference is crucial for universality (Remark 1). Compared to Morita's definition [20, Definition 2.1], the above Definition 2 does not have separate (un)conditional jump instructions. As is shown in Section 3, in the reversible setting this difference allows for a decision procedure (Theorem 21) for the corresponding halting problem.

▶ **Example 3.** Consider $\mathcal{M} = [\mathtt{dec}_0\, 4, \mathtt{inc}_0, \mathtt{dec}_0\, 0]$. The configuration $(0, (a, b))$ for $a, b \in \mathbb{N}$ is terminating in $\mathcal{M}$ iff $a > 0$, as is shown below:

$$(0, (0, b)) \overset{\mathtt{dec}_0\,4}{\longrightarrow}_{\mathcal{M}} (1, (0, b)) \overset{\mathtt{inc}_0}{\longrightarrow}_{\mathcal{M}} (2, (1, b)) \overset{\mathtt{dec}_0\,0}{\longrightarrow}_{\mathcal{M}} (0, (0, b)) \overset{\mathtt{dec}_0\,4}{\longrightarrow}_{\mathcal{M}} \cdots$$

$$(0, (a + 1, b)) \overset{\mathtt{dec}_0\,4}{\longrightarrow}_{\mathcal{M}} (4, (a, b)) \text{ halts}$$

By definition, the step relation $(\longrightarrow_{\mathcal{M}})$ for $\mathcal{M}$ is functional, and we can define a computable partial step function.

▶ **Definition 4** (Partial Step Function, Run)**.** For a machine $\mathcal{M}$, the *partial step function* is given by $\mathcal{M}(x) = y$ if $x \longrightarrow_{\mathcal{M}} y$.

A *run* in $\mathcal{M}$ starting from a configuration x is the (potentially infinite) sequence $x, \mathcal{M}(x), \mathcal{M}^2(x), \mathcal{M}^3(x), \ldots$.

Famously, the halting problem for two-counter machines (Problem 5) is undecidable.

▶ **Problem 5** (Two-Counter Machine Halting)**.** Given a two-counter machine $\mathcal{M}$ and a configuration x, is x terminating in $\mathcal{M}$?

Minsky's universality proof for program-machines carries over to CM2, resulting in the undecidability of the corresponding halting problem (Theorem 6). Most importantly, a conditional jump at position p to program index q if counter c is zero can be simulated by the instructions $[\mathtt{dec}_c\,(p + 3), \mathtt{inc}_c, \mathtt{dec}_c\, q]$.

▶ **Theorem 6** ([19, Section 11.5] and [19, Theorem 14.1-1])**.** Two-counter machine halting (Problem 5) is undecidable.

▶ **Remark 7.** Theorem 6 is mechanized by Forster et al. [7], as part of the Coq library of undecidability proofs.

3 Reversible Machines

In this section we consider *reversible* two-counter machines (also called *backward deterministic*). That is, machines with a right-unique step relation (or, injective step function). We show that for CM2 *reversibility* (is a given machine reversible?) and *reversible halting* (is a given configuration terminating in a given reversible machine?) are decidable problems. This contrasts the negative result by Morita [20, Theorem 4.2] for a different two-counter machine model with a richer instruction set. The positive result is unexpected because CM2 and Morita's machine model can simulate one another preserving determinism (as opposed to backwards-determinism).

▶ **Definition 8** (Reversible Machine)**.** A machine $\mathcal{M}$ is *reversible* if for any configurations x, y, z such that $x \longrightarrow_{\mathcal{M}} z$ and $y \longrightarrow_{\mathcal{M}} z$ we have $x = y$.

▶ **Example 9.** Consider $\mathcal{M} = [\mathtt{dec}_0\, 4, \mathtt{inc}_0, \mathtt{dec}_0\, 0]$ from Example 3. The machine $\mathcal{M}$ is reversible because $(\longrightarrow_{\mathcal{M}})$, fully given below, is right-unique.

$$(0, (0, b)) \xrightarrow{\mathtt{dec}_0 4}_{\mathcal{M}} (1, (0, b)) \qquad (0, (a+1, b)) \xrightarrow{\mathtt{dec}_0 4}_{\mathcal{M}} (4, (a, b))$$

$$(1, (a, b)) \xrightarrow{\mathtt{inc}_0}_{\mathcal{M}} (2, (a+1, b))$$

$$(2, (0, b)) \xrightarrow{\mathtt{dec}_0 0}_{\mathcal{M}} (3, (0, b)) \qquad (2, (a+1, b)) \xrightarrow{\mathtt{dec}_0 0}_{\mathcal{M}} (0, (a, b))$$

▶ **Remark 10.** A reversible machine $\mathcal{M}$ cannot contain both instructions $\mathtt{dec}_0\, q$ (at position p_1) and $\mathtt{dec}_1\, q$ (at position p_2). Otherwise, the transitions $(p_1, (1, 0)) \xrightarrow{\mathtt{dec}_0 q}_{\mathcal{M}} (q, (0, 0))$ and $(p_2, (0, 1)) \xrightarrow{\mathtt{dec}_1 q}_{\mathcal{M}} (q, (0, 0))$ contradict reversibility of $\mathcal{M}$.

▶ **Remark 11.** Morita gives a different, syntactic definition of reversibility (no *range overlap* [20, Definition 2.3]), specific to the underlying machine model. In fact, Morita's characterization is stronger than right-uniqueness of the step relation (cf. Definition 8), because it also takes into account the instruction used. Therefore, Morita's negative result [20, Theorem 4.2] holds a fortiori, when reversibility is defined by right-uniqueness of the step relation (cf. Definition 3). Compared to the positive result (Theorem 21) in the present work, the richer instruction set is the main contributing factor to undecidability, and not the particular definition of reversibility.

Before we consider the corresponding halting problem, let us ensure that we can decide membership (Corollary 14) in the class of reversible machines.

▶ **Problem 12** (Two-Counter Machine Reversibility)**.** Given a two-counter machine $\mathcal{M}$, is $\mathcal{M}$ reversible?

The following Lemma 13 bounds the set of configurations, which suffices to characterize reversibility.

▶ **Lemma 13.** For a machine $\mathcal{M}$ let $T_{\mathcal{M}} = \{(i, (a, b)) \mid i < |\mathcal{M}|, a \leq 2, b \leq 2\}$. If for all $x, y \in T_{\mathcal{M}}$ such that $\mathcal{M}(x) = \mathcal{M}(y)$ we have $x = y$, then $\mathcal{M}$ is reversible.

Proof. Given a machine $\mathcal{M}$, the step relation $(\longrightarrow_{\mathcal{M}})$ only depends on the current program index (bounded by $|\mathcal{M}|$) and on whether the current counters are positive or zero. Additionally, in one step the counters may only increase/decrease by one. Assume for some configurations x, y, z such that $x \neq y$ we have $x \longrightarrow_{\mathcal{M}} z$ and $y \longrightarrow_{\mathcal{M}} z$. By exhaustive case analysis on the instruction taken, we can sufficiently decrease the counter values in x, y, z, constructing distinct configurations $x', y' \in T_{\mathcal{M}}$ such that $\mathcal{M}(x') = \mathcal{M}(y')$. ◀

As an immediate consequence of the above Lemma 13, reversibility is decidable (in polynomial time).

▶ **Corollary 14.** Two-counter machine reversibility (Problem 12) is decidable.

Reversible machine halting (Problem 15) is the restriction of the halting problem to reversible machines.

▶ **Problem 15** (Two-Counter Reversible Machine Halting)**.** Given a reversible two-counter machine $\mathcal{M}$ and a configuration x, is x terminating in $\mathcal{M}$?

Morita shows that for a two-counter machine model, which is different from CM2, reversible machine halting is undecidable [20, Theorem 4.2]. Surprisingly, this result does not translate to CM2. The main difference is that in Morita's case the richer instruction set allows for reversible nested loops, whereas CM2 does not. In the remainder of this section we give a decision procedure for reversible machine halting for CM2.

The following Example 16 shows that for the reversible machine from Examples 3 and 9 it does not suffice to inspect positivity of the counters in a given configuration to decide termination.

▶ **Example 16.** Consider the reversible machine $\mathcal{M} = [\mathtt{dec}_0\, 4, \mathtt{inc}_0, \mathtt{dec}_0\, 0]$ from Example 3. The configurations $(2,(0,0))$ and $(2,(2,0))$ are terminating, but $(2,(1,0))$ is not, because

$$(2,(0,0)) \xrightarrow{\mathtt{dec}_0\,0}_{\mathcal{M}} (3,(0,0)) \text{ halts} \qquad \text{(terminating)}$$

$$(2,(1,0)) \xrightarrow{\mathtt{dec}_0\,0}_{\mathcal{M}} (0,(0,0)) \xrightarrow{\mathtt{dec}_0\,4}_{\mathcal{M}} (1,(0,0)) \xrightarrow{\mathtt{inc}_0}_{\mathcal{M}} (2,(1,0)) \xrightarrow{\mathtt{dec}_0\,0}_{\mathcal{M}} \ldots \qquad \text{(non-terminating)}$$

$$(2,(2,0)) \xrightarrow{\mathtt{dec}_0\,0}_{\mathcal{M}} (0,(1,0)) \xrightarrow{\mathtt{dec}_0\,4}_{\mathcal{M}} (4,(0,0)) \text{ halts} \qquad \text{(terminating)}$$

However, in the above Example 16, considering the program index 0, it *does* suffice to inspect positivity of the counters to decide termination. In fact, this is systematic for reversible CM2, which is the key insight (Lemma 18) to decide the reversible halting problem (Theorem 21). Additionally, any run of a reversible CM2 will eventually halt or reach the program index 0 (Lemma 17).

▶ **Lemma 17.** Let $\mathcal{M}$ be a reversible machine. Given a configuration x we can compute a configuration $(p,(a,b))$ such that $x \longrightarrow^*_{\mathcal{M}} (p,(a,b))$, and $p = 0$ or $p \geq |\mathcal{M}|$.

Proof. For $x = (p,(a,b))$ by induction on $\max\{(|\mathcal{M}| - p), 0\}$. The only interesting case is when $\mathtt{dec}_c\, q$ is the p-th instruction of $\mathcal{M}$ and $0 < q < |\mathcal{M}|$. In this case, by exhaustive case analysis on the $(q-1)$-th instruction of $|\mathcal{M}|$, we can contradict reversibility of $\mathcal{M}$. ◀

Let us consider the partitioning $\mathcal{T} = \{T_{(0,0)}, T_{(0,1)}, T_{(1,0)}, T_{(1,1)}\}$ of configurations at program index zero with respect to positivity of counters, where

$$T_{(0,0)} = \{(0,(0,0))\}$$
$$T_{(0,1)} = \{(0,(0,b)) \mid 1 \leq b\}$$
$$T_{(1,0)} = \{(0,(a,0)) \mid 1 \leq a\}$$
$$T_{(1,1)} = \{(0,(a,b)) \mid 1 \leq a, 1 \leq b\}$$

The following Lemma 18 shows that for a reversible machine in each partition all configurations exhibit uniform behavior.

▶ **Lemma 18.** Let $\mathcal{M}$ be a reversible machine and let $T \in \mathcal{T}$. We can

1. show for all $x \in T$ that x terminates,
2. show for all $x \in T$ that x does not terminate, or
3. compute $T' \in \mathcal{T}$ such that for all $x \in T$ there is a $y \in T'$ such that $x \longrightarrow^+_{\mathcal{M}} y$.

Proof. Let $a_0, b_0 \in \{0,1\}$ and $T_{(a_0,b_0)} \in \mathcal{T}$. We compute the prefix (of length at most $|\mathcal{M}|$) of a run from $(0,(a_0,b_0))$ with increasing program indices. The run either halts or reaches the instruction $\mathtt{dec}_c\, 0$ such that the next configuration is $y = (0,(a',b'))$ (similarly to the proof of Lemma 17). The following case analysis provides an overview over the argument. The individual cases are by case analysis on the possible instructions taken.

Case $(a_0,b_0) = (0,0)$: For $T' = T_{(\min\{1,a'\},\min\{1,b'\})}$ we have $(0,(0,0)) \longrightarrow^+_{\mathcal{M}} y \in T'$.

Case $(a_0, b_0) = (0, 1)$: There are four cases
- **$1 \leq a'$ and $1 \leq b'$:** for $T' = T_{(1,1)}$ we have $(0, (0, b+1)) \longrightarrow^+_{\mathcal{M}} (0, (a', b+b')) \in T'$.
- **$a' = 0$ and $1 \leq b'$:** any configuration $(0, (0, b+1))$ is non-terminating.
- **$1 \leq a'$ and $b' = 0$:** we have $(0, (0, b+1)) \longrightarrow^+_{\mathcal{M}} (0, (a', b))$. By case analysis on b, taking Remark 10 into account, and depending on the uniform behavior of configurations in $T_{(1,1)}$, we have that any configuration $(0, (0, b+1))$ is terminating, or for $T' = T_{(1,0)}$ we have that $(0, (0, b+1)) \longrightarrow^+_{\mathcal{M}} (0, (a''+1, 0)) \in T'$ for some $a'' \in \mathbb{N}$.
- **$a' = 0$ and $b' = 0$:** for $T' = T_{(0,0)}$ we have $(0, (0, b+1)) \longrightarrow^+_{\mathcal{M}} (0, (0, 0)) \in T'$.

Case $(a_0, b_0) = (1, 0)$: Analogous to case $T = T_{(0,1)}$.

Case $(a_0, b_0) = (1, 1)$: There are three cases (the case $a' = 0 = b'$ is not possible).
- **$1 \leq a'$ and $1 \leq b'$:** for $T' = T_{(1,1)}$ we have $(0, (a+1, b+1)) \longrightarrow^+_{\mathcal{M}} (0, (a+a', b+b')) \in T'$.
- **$a' = 0$ and $1 \leq b'$:** for $T' = T_{(0,1)}$ we have $(0, (a+1, b+1)) \longrightarrow^+_{\mathcal{M}} (0, (0, b''+1)) \in T'$ for some $b'' \in \mathbb{N}$.
- **$1 \leq a'$ and $b' = 0$:** for $T' = T_{(1,0)}$ we have $(0, (a+1, b+1)) \longrightarrow^+_{\mathcal{M}} (0, (a''+1, 0)) \in T'$ for some $a'' \in \mathbb{N}$. ◀

The following Example 19 illustrates the uniform behavior of configurations in each partition in $\mathcal{T}$, as described in the above Lemma 18.

▶ **Example 19.** Consider the machine $\mathcal{M} = [\mathtt{inc}_1, \mathtt{dec}_0\, 5, \mathtt{inc}_0, \mathtt{dec}_0\, 0]$. Similarly to Example 9, $\mathcal{M}$ is reversible. The uniform behavior of configurations in each partition in $\mathcal{T}$ is as follows.
- For $(0, (0, 0)) \in T_{(0,0)}$ we have $(0, (0, 0)) \longrightarrow^+_{\mathcal{M}} (0, (0, 1)) \in T_{(0,1)}$.
- For all $(0, (0, b+1)) \in T_{(0,1)}$ we have $(0, (0, b+1)) \longrightarrow^+_{\mathcal{M}} (0, (0, b+2)) \in T_{(0,1)}$.
- For all $(0, (a+1, 0)) \in T_{(1,0)}$ we have that $(0, (a+1, 0))$ terminates after 2 steps.
- For all $(0, (a+1, b+1)) \in T_{(1,1)}$ we have that $(0, (a+1, b+1))$ terminates after 2 steps.

▶ **Remark 20.** Morita's reversible universal machine construction [20, Proof of Theorem 4.1] requires computational distinction between odd and even counter values. Therefore, for Morita's richer instruction set there is a reversible machine for which the configurations $(0, (2 \cdot a, 0))$ are terminating and configurations $(0, (2 \cdot a+1, 0))$ are not terminating. In contrast, by Lemma 18 there is no such reversible CM2. For instance, the configurations $(0, (2, 0))$ and $(0, (3, 0))$ are both members of the partition $T_{(1,0)}$, and expose uniform termination behavior for any reversible CM2.

As a result of Lemma 18, termination of configurations with program index 0 is characterized by a finite state automaton with the four states $\mathcal{T}$. Combined with Lemma 17, we obtain a decision procedure for reversible machine halting (Theorem 21).

▶ **Theorem 21.** Two-counter reversible machine halting (Problem 15) is decidable.

Proof. Given a reversible machine $\mathcal{M}$ and a configuration x, by Lemma 17 compute the configuration $(p, (a, b))$ such that $x \longrightarrow^*_{\mathcal{M}} (p, (a, b))$ and $p = 0$ or $p \geq |\mathcal{M}|$. In case $p \geq |\mathcal{M}|$ we have that $(p, (a, b))$ halts, and therefore x is terminating. If $p = 0$, then for $a_0 = \min\{1, a\}$ and $b_0 = \min\{1, b\}$ we have $(p, (a, b)) \in T_{(a_0,b_0)}$. Using Lemma 18, compute the finite state automaton with states $\mathcal{T}$, where the initial state is $T_{(a_0,b_0)}$, the accepting states satisfy (18.1), and the transition function corresponds to (18.3). Termination of $x \in T_{(a_0,b_0)}$ corresponds to reachability of any accepting state in the constructed finite automaton, which is decidable. ◀

▶ **Remark 22.** The proof of Theorem 21 entails a *polynomial time* decision procedure for reversible halting. Notably, in order to construct the finite state automaton with states $\mathcal{T}$ for a given machine $\mathcal{M}$, it suffices to inspect a constant number of runs of length at most $|\mathcal{M}|$.

4 Boundedness

In this section we consider boundedness properties of two-counter machines. First, we recall that *total boundedness* (does every run eventually halt or enter a configuration cycle?) is undecidable [14, Theorem 8], which also holds for the machine model at hand (Theorem 28). Second, for *uniform boundedness* (is there a uniform bound on the number of reachable configurations?) we contribute a decision procedure (Theorem 40). Techniques presented in this section are not specific to CM2 and extend to other two-counter machine models.

If a run starting from a configuration x in a machine $\mathcal{M}$ halts or enters a configuration cycle, then we can bound the number of reachable configurations from x in $\mathcal{M}$ (Definition 23).

▶ **Definition 23** (Bound). An $n \in \mathbb{N}$ *bounds* a configuration x in a machine $\mathcal{M}$ if we have $|\{y \mid x \longrightarrow_{\mathcal{M}}^{*} y\}| \leq n$.

For a *totally bounded* machine every configuration is bounded. In other words, a totally bounded machine has no aperiodic, non-terminating runs.

▶ **Definition 24** (Totally Bounded Machine). A machine $\mathcal{M}$ is *totally bounded* if for all configurations x there exists an $n \in \mathbb{N}$ such that n bounds x in $\mathcal{M}$.

▶ **Remark 25.** Every (possibly infinite) run of a totally bounded machine can be fully described by a finite prefix. This renders key properties such as reachability and termination decidable, and is useful for model-checking.

▶ **Example 26.** Consider $\mathcal{M} = [\mathtt{dec}_0\, 0, \mathtt{inc}_0, \mathtt{dec}_0\, 0]$. Configurations $(0, (a, b))$ are bounded by $a + 3$ in $\mathcal{M}$ because

$$(0,(a,b)) \longrightarrow_{\mathcal{M}}^{a} (0,(0,b)) \xrightarrow{\mathtt{dec}_0\, 0}_{\mathcal{M}} (1,(0,b)) \xrightarrow{\mathtt{inc}_0}_{\mathcal{M}} (2,(1,b)) \xrightarrow{\mathtt{dec}_0\, 0}_{\mathcal{M}} (0,(0,b)) \longrightarrow_{\mathcal{M}} \dots$$

Overall, the machine $\mathcal{M}$ is totally bounded with the following bounds

$(0,(a,b))$	bounded by $a+3$
$(1,(a,b)) \xrightarrow{\mathtt{inc}_0}_{\mathcal{M}} (2,(a+1,b)) \xrightarrow{\mathtt{dec}_0\, 0}_{\mathcal{M}} (0,(a,b))$	bounded by $a+5$
$(2,(0,b)) \xrightarrow{\mathtt{dec}_0\, 0}_{\mathcal{M}} (3,(0,b))$ halts	bounded by 2
$(2,(a+1,b)) \xrightarrow{\mathtt{dec}_0\, 0}_{\mathcal{M}} (0,(a,b))$	bounded by $a+4$
$(p+3,(a,b))$ halts	bounded by 1

The negative result by Kuzmin and Chalyy [14, Theorem 8] for two-counter machine total boundedness (Problem 27) translates to the machine model at hand (Theorem 28).

▶ **Problem 27** (Two-Counter Machine Total Boundedness). Given a two-counter machine $\mathcal{M}$, is $\mathcal{M}$ totally bounded?

▶ **Theorem 28** ([14, Theorem 8]). Two-counter machine total boundedness (Problem 27) is undecidable.

In contrast to a totally bounded machine, for a *uniformly bounded* machine the bound on the number of reachable configurations does not depend on the starting configuration.

▶ **Definition 29** (Uniform Bound). An $n \in \mathbb{N}$ *uniformly bounds* a machine $\mathcal{M}$ if n bounds all configurations x in $\mathcal{M}$.

▶ **Definition 30** (Uniformly Bounded Machine)**.** A machine $\mathcal{M}$ is *uniformly bounded* if there exists a uniform bound n of $\mathcal{M}$.

Intuitively, a uniformly bounded machine operates in bounded space, given by the particular uniform bound. The following Example 31 shows that uniform boundedness is strictly stronger than total boundedness.

▶ **Example 31.** Consider the totally bounded machine $\mathcal{M} = [\mathtt{dec}_0\, 0, \mathtt{inc}_0, \mathtt{dec}_0\, 0]$ from Example 26. There is no uniform bound for $\mathcal{M}$ because for any $n \in \mathbb{N}$ from the configuration $(0, (n, 0))$ more than n distinct configurations are reachable.

▶ **Remark 32.** It is surprising that any model of computation can have computationally interesting, uniformly bounded machines. For example, such machines are not capable of processing an arbitrarily large input. However, for Turing machines, relying on the ingenious technique developed by Hooper [10] one can reduce Turing machine halting to a uniform boundedness problem for stack machines [2].

▶ **Problem 33** (Two-Counter Machine Uniform Boundedness)**.** Given a two-counter machine $\mathcal{M}$, is $\mathcal{M}$ uniformly bounded?

In the remainder of this section we give a decision procedure for two-counter machine uniform boundedness.

First, we characterize whether n bounds a configuration x by inspection of at most the first n steps in a run from x.

▶ **Fact 34.** An $n \in \mathbb{N}$ bounds a configuration x in a machine $\mathcal{M}$ iff either $\mathcal{M}^n(x)$ is undefined or $\mathcal{M}^n(x) = \mathcal{M}^m(x)$ for some $m < n$.

The above Fact 34 entails a decision procedure to determine whether n bounds x in $\mathcal{M}$.

▶ **Corollary 35.** Given a machine $\mathcal{M}$, a configuration x, and an $n \in \mathbb{N}$, it is decidable whether n bounds x in $\mathcal{M}$.

Second, we characterize whether n bounds a machine $\mathcal{M}$ by inspection of the finitely many configurations with counters at most n.

▶ **Lemma 36.** Let $\mathcal{M}$ be a machine and let $n \in \mathbb{N}$. If for all $p \leq |\mathcal{M}|$, $a \leq n$, and $b \leq n$ we have that n bounds $(p, (a, b))$ in $\mathcal{M}$, then n uniformly bounds $\mathcal{M}$.

Proof. By Fact 34, for any configuration x it suffices to inspect the first n steps of any run from x to decide whether n bounds x. Since at each step the counter values change by at most one, configurations with counter values of at least n behave uniformly after at most n steps with respect to halting or entering a configuration cycle. ◀

As a result of the above Lemma 36 and Corollary 35, it is decidable whether n bounds $\mathcal{M}$.

▶ **Corollary 37.** Given a machine $\mathcal{M}$ and an $n \in \mathbb{N}$, it is decidable whether n uniformly bounds $\mathcal{M}$.

Third, we give a sufficient condition for aperiodic, arbitrary long runs (Lemma 38). A machine satisfying this condition cannot be uniformly bounded. Intuitively, the condition captures repeatable program index cycles for which at least one counter value changes. Considering counter values which are at least the cycle length, larger counter values exhibit identical control flow.

▶ **Lemma 38.** Let $\mathcal{M}$ be a machine, let $k \in \mathbb{N}$, and let $(p, (a_1, b_1)), (p, (a_2, b_2))$ be configurations such that $(p, (a_1, b_1)) \longrightarrow^k_{\mathcal{M}} (p, (a_2, b_2))$. If all of the following conditions hold, then $\mathcal{M}$ is not uniformly bounded.

(1) $k \leq a_1$ or $a_1 = a_2$ (2) $k \leq b_1$ or $b_1 = b_2$ (3) $a_1 \neq a_2$ or $b_1 \neq b_2$

Proof. Assume that some n uniformly bounds $\mathcal{M}$. Consider the case $k \leq a_1 \neq a_2$ and $b_1 = b_2 = b$ (the other cases are analogous). In the first $k - 1$ steps from $(p, (a_1, b_1))$ the first counter is positive. Therefore, from the configuration $(p, (a_1 + n \cdot a_1, b))$ there are more than n distinct reachable configurations:

$$\begin{aligned}(p, (a_1 + n \cdot a_1, b)) &\longrightarrow^k_{\mathcal{M}} (p, (a_1 + (n-1) \cdot a_1 + 1 \cdot a_2, b)) \\ &\longrightarrow^k_{\mathcal{M}} (p, (a_1 + (n-2) \cdot a_1 + 2 \cdot a_2, b)) \\ &\longrightarrow^k_{\mathcal{M}} \ldots \longrightarrow^k_{\mathcal{M}} (p, (a_1 + n \cdot a_2, b))\end{aligned}$$

This contradicts the uniform bound n of $\mathcal{M}$. ◀

Fourth, we characterize whether $\mathcal{M}$ is uniformly bounded by inspection of the particular bound $(|\mathcal{M}| + 1)^5$. By the pigeonhole principle this upper bound covers sufficiently many configurations to exploit the negative condition (Lemma 38) for uniform boundedness.

▶ **Lemma 39.** If a machine $\mathcal{M}$ is uniformly bounded, then $(|\mathcal{M}| + 1)^5$ uniformly bounds $\mathcal{M}$.

Proof. For an arbitrary configuration x, consider the first at most $(|\mathcal{M}| + 1)^5$ steps of a run in $\mathcal{M}$ from x. If the run halts or enters a configuration cycle, then $(|\mathcal{M}| + 1)^5$ bounds x in $\mathcal{M}$. Otherwise, we contradict that $\mathcal{M}$ is uniformly bounded as follows. In this case, the considered $(|\mathcal{M}| + 1)^5$ configurations are distinct (and defined).

Let $l = |\mathcal{M}| \cdot (|\mathcal{M}| + 1)$. By the pigeonhole principle, in the first $|\mathcal{M}| \cdot l^2$ steps, we encounter a configuration $x_1 = (p_1, (a_1, b_1))$ such that $l \leq a_1$ or $l \leq b_1$. Consider $l \leq a_1$ (the other case is analogous). For the the next $|\mathcal{M}|^2$ steps the first counter is at least $|\mathcal{M}|$, and there are two cases.

Case 1: We encounter a configuration $x_2 = (p_2, (a_2, b_2))$ such that $|\mathcal{M}| \leq a_2$ and $|\mathcal{M}| \leq b_2$. By the pigeonhole principle, in the next $|\mathcal{M}|$ steps, we necessarily encounter configurations $x_3 = (p, (a_3, b_3))$ and $x_4 = (p, (a_4, b_4))$ such that $x_3 \longrightarrow^k_{\mathcal{M}} x_4$, $k \leq a_3$, $k \leq b_3$, and $x_3 \neq x_4$. This contradicts uniform boundedness of $\mathcal{M}$ by Lemma 38.

Case 2: All configurations are such that the second counter is less than $|\mathcal{M}|$. By the pigeonhole principle, we encounter configurations $x'_3 = (p', (a'_3, b'))$ and $x'_4 = (p', (a'_4, b'))$ such that $x'_3 \longrightarrow^k_{\mathcal{M}} x'_4$ and $k \leq a'_3 \neq a'_4$. This contradicts uniform boundedness of $\mathcal{M}$ by Lemma 38. ◀

Finally, relying on the above Lemma 39 and Corollary 37, we give a decision procedure for uniform boundedness.

▶ **Theorem 40.** Two-counter machine uniform boundedness (Problem 27) is decidable.

Proof. Given a machine $\mathcal{M}$, by Lemma 39 it suffices to decide whether $(|\mathcal{M}| + 1)^5$ uniformly bounds $\mathcal{M}$, which is decidable by Corollary 37. ◀

▶ **Remark 41.** The proof of Theorem 40 entails a *polynomial time* decision procedure for uniform boundedness. In particular, given a machine $\mathcal{M}$ we inspect the potential uniform bound $n = (|\mathcal{M}| + 1)^5$ (cf. Lemma 39). That is, we inspect whether n bounds configurations $(p, (a, b))$ such that $p \leq |\mathcal{M}|$, $a \leq n$, and $b \leq n$ (cf. Lemma 36). Each of these $|\mathcal{M}| \cdot n^2$ configurations halts or enters a configuration cycle in the first n steps in $\mathcal{M}$ (cf. Fact 34) iff $\mathcal{M}$ is uniformly bounded.

5 Mortality

In this section we consider mortality properties of two-counter machines. First, we recall that *total mortality* (does every run eventually halt?) is undecidable [10, Part VI.7], which also holds for the machine model at hand (Theorem 45). Second, for *uniform mortality* (is there a uniform bound on the number of steps from any configuration?) we give a decision procedure (Theorem 49).

▶ **Definition 42** (Totally Mortal Machine)**.** A machine $\mathcal{M}$ is *totally mortal* if all configurations x are terminating in $\mathcal{M}$.

Total mortality is strictly stronger than total boundedness (Example 43).

▶ **Example 43.** Consider the totally bounded machine $\mathcal{M} = [\mathtt{dec}_0\, 0, \mathtt{inc}_0, \mathtt{dec}_0\, 0]$ from Example 26. The machine $\mathcal{M}$ is not totally mortal because of the non-terminating run

$$(0,(0,0)) \xrightarrow{\mathtt{dec}_0\, 0}_{\mathcal{M}} (1,(0,0)) \xrightarrow{\mathtt{inc}_0}_{\mathcal{M}} (2,(1,0)) \xrightarrow{\mathtt{dec}_0\, 0}_{\mathcal{M}} (0,(0,0)) \longrightarrow_{\mathcal{M}} \dots$$

The original negative result by Hooper [10, Part VI.7] for two-counter machine total mortality (Problem 44) translates to the machine model at hand (Theorem 45).

▶ **Problem 44** (Two-Counter Machine Total Mortality)**.** Given a two-counter machine $\mathcal{M}$, is $\mathcal{M}$ totally mortal?

▶ **Theorem 45** ([10, Part VI.7])**.** Two-counter machine total mortality (Problem 44) is undecidable.

▶ **Remark 46.** The negative result for *reversible* two-counter machine total mortality [12, Theorem 1] does *not* hold for CM2. By Lemma 18, it suffices to inspect reachability of accepting states in a computable finite state automaton, which is decidable.

In a *uniformly mortal* machine there is a uniform bound on the number of steps after which every run halts.

▶ **Definition 47** (Uniformly Mortal Machine)**.** A machine $\mathcal{M}$ is *uniformly mortal* if there exists an $n \in \mathbb{N}$ such that for any configuration x we have that $\mathcal{M}^n(x)$ is undefined.

Kari and Ollinger sketch a decision procedure [12, Theorem 2] for uniform mortality (Problem 48). In the remainder of this section we give an alternative decision procedure (Theorem 49), based on the decision procedure for uniform boundedness.

▶ **Problem 48** (Two-Counter Machine Uniform Mortality)**.** Given a two-counter machine $\mathcal{M}$, is $\mathcal{M}$ uniformly mortal?

▶ **Theorem 49.** Two-counter machine uniform mortality (Problem 48) is decidable.

Proof. Given a machine $\mathcal{M}$, decide whether $\mathcal{M}$ is uniformly bounded. If not, then $\mathcal{M}$ is not uniformly mortal. Otherwise, $n = (|\mathcal{M}| + 1)^5$ uniformly bounds $\mathcal{M}$ by Lemma 39. It suffices to decide whether n bounds the maximal number of steps from any configuration. Configurations with counter values at least n behave uniformly for the first n steps. Therefore, it suffices to inspect the finitely many configurations $(p,(a,b))$ such that $p \leq |\mathcal{M}|$, $a \leq n$, $b \leq n$. All such configurations halt after at most n steps iff $\mathcal{M}$ is uniformly mortal. ◀

▶ **Remark 50.** The proof of Theorem 49 entails a *polynomial time* decision procedure for uniform mortality. In particular, given a machine $\mathcal{M}$ we inspect whether $n = (|\mathcal{M}| + 1)^5$ bounds the number of steps from configurations $(p,(a,b))$ such that $p \leq |\mathcal{M}|$, $a \leq n$, and $b \leq n$. Each of these $|\mathcal{M}| \cdot n^2$ configurations halts in at most n steps iff $\mathcal{M}$ is uniformly mortal.

6 Mechanization

At the level of human intuition, the decidability results for reversible halting (Theorem 21), uniform boundedness (Theorem 40), and uniform mortality (Theorem 49) are uncomplicated. However, the necessary verification of the individual results in full-detail, often by nested case analysis, is laborious and (without computer assistance) error-prone.

A mechanization of the presented results using a proof assistant constitutes a rigorous, mechanically verifiable correctness proof. Using the Coq proof assistant [27] in particular has several benefits. First, as argued by Forster [3, Chapter 2], Coq is well-suited for positive and negative computability results because of its separate impredicative universe of propositions, besides a computational type hierarchy. This separation allows for a distinction between computational and non-computational aspects of computability results. In addition, any function implemented in axiom-free Coq is, by design, total and computable. Second, the Coq library of undecidability proofs [8] is a readily available uniform framework to mechanize computability results. The present work heavily relies on the existing infrastructure for two-counter machines provided by the library. Third, using the `Extraction` framework [17] one can extract effective implementations of the individual decision procedures.

The library defines[3] decidability of decision problems as follows.

```
Definition reflects (b : bool) (p : Prop) := p <-> b = true.

Definition decider {X} (f : X -> bool) (P : X -> Prop) : Prop :=
  forall x, reflects (f x) (P x).

Definition decidable {X} (P : X -> Prop) : Prop :=
  exists f : X -> bool, decider f P.
```

In particular, a problem `P : X -> Prop` on the domain `X` is decidable, if there exists a Boolean function `f : X -> bool` such that for all `x` in `X` we have that `P x` holds iff `f x = true` (cf. *small scale reflection* [9]).

In general, a propositional existence proof of a computable decision procedure can rely on non-constructive principles, such as the principle of excluded middle. In the present work we mechanize the individual decision procedures in axiom-free Coq, which constitutes the strongest result with respect to constructive mathematics.

The library contains[4] the following definition of two-counter machines (cf. Definition 2).

```
Definition Config : Set := nat * (nat * nat).

Definition state (x: Config) : nat := fst x.
Definition value1 (x: Config) : nat := fst (snd x).
Definition value2 (x: Config) : nat := snd (snd x).

Inductive Instruction : Set :=
  | inc : bool -> Instruction
  | dec : bool -> nat -> Instruction.

Definition Cm2 : Set := list Instruction.

Definition step (M: Cm2) (x: Config) : option Config :=
  [...]

Definition steps (M: Cm2) (k: nat) (x: Config) : option Config :=
  Nat.iter k (obind (step M)) (Some x).
```

[3] `theories/Synthetic/Definitions.v`
[4] `theories/CounterMachines/CM2.v`

In the above, a two-counter machine of type `Cm2` with the configuration space `nat * (nat * nat)` is a list of instructions of shape either `(inc false)` for $\mathtt{inc}_0$, `(inc true)` for $\mathtt{inc}_1$, `(dec false q)` for $\mathtt{dec}_0\, q$, or `(dec true q)` for $\mathtt{dec}_1\, q$. The function `step : Cm2 -> Config -> option Config`, mechanizes the two-counter machine partial step function (Definition 4). For exactly the halting configurations `x : Config` we have `step M x = None`. The iterated step function is `steps : Cm2 -> nat -> Config -> option Config`. A prominent result[5] in the library is the undecidability of the halting problem for two-counter machines.

The remainder of this section outlines the mechanization of the decision procedures contributed by the present work to the Coq library of undecidability proofs.

Reversible Halting

The following predicate `CM2_REV_HALT` mechanizes the reversible halting problem for two-counter machines (Problem 15).

```
Definition terminating (M: Cm2) (x: Config) :=
  exists k, steps M k x = None.

Definition reversible (M : Cm2) : Prop :=
  forall x y z, step M x = Some z -> step M y = Some z -> x = y.

Definition CM2_REV_HALT : { M: Cm2 | reversible M } * Config -> Prop :=
  fun '((exist _ M _), x) => terminating M x.
```

In particular, given[6] a two-counter machine `M : Cm2`, a proof that `M` is `reversible` (`step` is injective), and a configuration `x : Config`, is there a `k : nat` such that `M` halts after at most `k` steps starting from configuration `x`?

The decision procedure `decide : { M: Cm2 | reversible M } * Config -> bool` for the predicate `CM2_REV_HALT` is mechanized in `theories/CounterMachines/Deciders/CM2_REV_HALT_dec.v` with the corresponding correctness proof `decide_spec : decider decide CM2_REV_HALT`. Notably, the key Lemma 18 for the construction of a finite state automaton to decide reversible halting is mechanized as follows (`RZ` mechanizes membership in the same partition in $\mathcal{T}$).

```
Lemma uniform_transition ab :
  In ab representatives ->
  (forall a'b', RZ ab a'b' -> terminating (0, a'b')) +
  (forall a'b', RZ ab a'b' -> non_terminating (0, a'b')) +
  (* uniform transition *)
  {v | In v representatives /\
    (forall a'b', RZ ab a'b' -> exists w, RZ v w /\
      reaches_plus (0, a'b') (0, w)) }.
```

In order to implement a computable decision procedure, it is important to use computational disjunction (`+`) and the dependent pair `{ v | In v representatives /\ ... }`. The corresponding propositional counterparts (`\/`) and (`exists v, In v representatives /\ ...`) do not suffice.

The overall mechanization spans approximately 1000 LOC. It relies heavily on the proof automation tactic `lia` for linear integer arithmetic.

[5] `theories/CounterMachines/CM2_undec.v`

[6] The syntax `'((exist _ M _), x)` matches a member of `{ M: Cm2 | reversible M } * Config`, and binds the given machine `M : Cm2` and the given configuration `x : Config`.

Uniform Boundedness

The following predicate `CM2_UBOUNDED` mechanizes uniform boundedness for two-counter machines (Problem 33).

```
Definition reaches (M: Cm2) (x y: Config) :=
  exists k, steps M k x = Some y.

Definition bounded (M: Cm2) (k: nat) (x: Config) : Prop :=
  exists (L: list Config), (length L <= k) /\
    (forall (y: Config), reaches M x y -> In y L).

Definition uniformly_bounded (M: Cm2) : Prop :=
  exists k, forall x, bounded M k x.

Definition CM2_UBOUNDED : Cm2 -> Prop :=
  fun M => uniformly_bounded M.
```

In particular, given a two-counter machine `M : Cm2`, is there a bound `k : nat` such that for all configurations `x : Config` the reachable configurations from `x` are bounded by a list `L : list Config` of length at most `k`?

The decision procedure `decide : Cm2 -> bool` for the predicate `CM2_UBOUNDED` is mechanized in `theories/CounterMachines/Deciders/CM2_UBOUNDED_dec.v` with the corresponding correctness proof `decide_spec : decider decide CM2_UBOUNDED`. Notably, the key Lemma 39 which provides an uniform upper bound is mechanized as follows (where `l` is `length M`).

```
Lemma bound_on_uniform_bound : uniformly_bounded M ->
  forall x, bounded M ((l+1)*(l+1)*(l+1)*(l+1)*(l+1)) x.
```

The overall mechanization spans approximately 400 LOC. It relies on the following negative pigeonhole principle.

```
Lemma pigeonhole {X : Type} (L L' : list X) :
  incl L L' -> length L' < length L -> not (NoDup L).
```

In particular, given two lists `L` and `L'`, if each element of `L` is in the strictly shorter list `L'`, then `L` is not duplicate-free.

Uniform Mortality

The following predicate `CM2_UMORTAL` mechanizes uniform mortality for two-counter machines (Problem 48).

```
Definition mortal (M: Cm2) (k: nat) (x: Config) : Prop :=
  steps M k x = None.

Definition uniformly_mortal (M: Cm2) : Prop :=
  exists k, forall x, mortal M k x.

Definition CM2_UMORTAL : Cm2 -> Prop :=
  fun M => uniformly_mortal M.
```

In particular, given a two-counter machine `M : Cm2`, is there a bound `k : nat` such that for all configurations `x : Config` the machine `M` starting from `x` halts after at most `k` steps?

The decision procedure `decide : Cm2 -> bool` for the predicate `CM2_UMORTAL` is mechanized in `theories/CounterMachines/Deciders/CM2_UMORTAL_dec.v` together with the corresponding correctness proof `decide_spec : decider decide CM2_UMORTAL`. The mechanization spans approximately 100 LOC and relies on the previously described mechanized decision procedure for uniform boundedness.

Extraction

Using the `Extraction` framework [17], effective implementations of the individual decision procedures (in the OCaml programming language) can be obtained from the provided mechanization. For example, the following code mechanizes Example 9 and Example 16.

```
From Undecidability Require Import CM2_REV_HALT_dec.
From Coq Require Import List Extraction.
Import CM2 ListNotations.

Definition M := [dec false 4; inc false; dec false 0].

Lemma HM : reversible M.
Proof.
  intros [[|[|[|[|xp]]]] [[|xa] xb]] [[|[|[|[|yp]]]] [[|ya] yb]] z.
  all: now cbn; congruence.
Qed.

Definition configs := [(2, (0, 0)); (2, (1, 0)); (2, (2, 0))].

Definition results := map (fun x => decide (exist _ M HM, x)) configs.
```

In the above, `M` mechanizes the machine $[\mathtt{dec}_0\, 4, \mathtt{inc}_0, \mathtt{dec}_0\, 0]$ from Example 9 and `HM` certifies reversibility of `M` by automated case analysis. Notably, the proof automation tactic `congruence` (implementing a congruence closure algorithm [23]) is well-suited for automated injectivity proofs. The list `configs` contains the three starting configurations $(2, (0, 0)), (2, (1, 0))$, and $(2, (2, 0))$ from Example 16, and the list `results` contains the corresponding halting decisions. Finally, using the command "`Recursive Extraction results.`" an OCaml implementation can be extracted. Upon execution, `results` returns the answers `true` for termination of the configurations $(2, (0, 0))$ and $(2, (2, 0))$, and `false` for termination of the configuration $(2, (1, 0))$, in agreement with Example 16. While still a toy example, this highlights both the suitability of the Coq proof assistant for (practical) computability theory, and the maturity of the underlying tool chain. Additionally, extraction to a widely-used programming language comes with toolchain, performance, and integration benefits for users outside of the proof assistant community.

7 Conclusion

The present work gives certified decision procedures for reversible halting (Theorem 21), uniform boundedness (Theorem 40), and uniform mortality (Theorem 49) for CM2, which is an established, computationally universal notion of two-counter machines.

The positive result for reversible halting contrasts universality of reversible two-counter machines [20] with a different, richer instruction set. The presented argument is by modeling relevant control flow of reversible CM2 by a finite state automaton. This renders the established instruction set of CM2 not computationally universal in a reversible setting.

The presented positive results for uniform boundedness (cf. [1, Remark 28]) and uniform mortality (cf. [12, Theorem 2]) provide insight into algorithmic complexity of the respective decision problems. In particular, the underlying arguments are based on a polynomial upper bound on the size of the relevant configuration space.

The described decision procedures are implemented and verified using the Coq proof assistant. Coq is well-suited for positive and negative computability results [3], which in practice culminates in a growing Coq library containing such results. By design, any function implemented in axiom-free Coq is computable and is equipped with a termination certificate. Therefore, both negative results (via computable reduction functions) and positive results

(via computable decision functions) can be presented and verified in a uniform framework. As added benefit, the complementary positive and negative results for individual problem classes (such as problems for two-counter machines) can rely on common infrastructure, avoiding code duplication. The library is well-maintained, which increases longevity of the contributed mechanization.

While the present work focuses on the positive results for CM2, it is desirable for the library to include mechanizations of the known negative results for total boundedness and total mortality.

Unfortunately, in contrast to computability, it is challenging to reason about time or space complexity of Coq code (cf. the active line of work by Kunze and Forster [4, 5, 6]). Therefore, certification of the polynomial time complexity (Remarks 22, 41, and 50) of the decision procedures given in the present work remains open.

The OCaml implementation of the individual decision procedures given by the `Extraction` framework is effective for the toy examples in the present work. However, it is neither efficient nor humanly readable due to the use of `ssreflect` proof tactic language [9]. This can be addressed by a more strict separation between decision procedures, termination proofs, and correctness proofs (cf. the *Braga method* [16] and the `Equations` framework [25]).

References

1 Andrej Dudenhefner. Undecidability of semi-unification on a napkin. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.9.

2 Andrej Dudenhefner. Constructive many-one reduction from the halting problem to semi-unification. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*, volume 216 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2022.18.

3 Yannick Forster. *Computability in Constructive Type Theory*. PhD thesis, Saarland University, 2021.

4 Yannick Forster and Fabian Kunze. A certifying extraction with time bounds from Coq to call-by-value λ-calculus. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*, pages 17:1–17:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITP.2019.17.

5 Yannick Forster, Fabian Kunze, and Marc Roth. The weak call-by-value λ-calculus is reasonable for both time and space. *Proc. ACM Program. Lang.*, 4(POPL):27:1–27:23, 2020. doi:10.1145/3371095.

6 Yannick Forster, Fabian Kunze, Gert Smolka, and Maximilian Wuttke. A mechanised proof of the time invariance thesis for the weak call-by-value λ-calculus. In Liron Cohen and Cezary Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPIcs*, pages 19:1–19:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITP.2021.19.

7 Yannick Forster and Dominique Larchey-Wendling. Certified undecidability of intuitionistic linear logic via binary stack machines and Minsky machines. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 104–117. ACM, 2019. doi:10.1145/3293880.3294096.

8 Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq Library of Undecidable Problems. In *The Sixth International Workshop on Coq for Programming Languages (CoqPL 2020)*, 2020. URL: `https://github.com/uds-psl/coq-library-undecidability`.

9 Georges Gonthier and Assia Mahboubi. An introduction to small scale reflection in Coq. *J. Formaliz. Reason.*, 3(2):95–152, 2010. `doi:10.6092/issn.1972-5787/1979`.

10 Philip K. Hooper. The undecidability of the Turing machine immortality problem. *J. Symb. Log.*, 31(2):219–234, 1966. `doi:10.2307/2269811`.

11 Sergiu Ivanov, Elisabeth Pelz, and Sergey Verlan. Small universal non-deterministic Petri nets with inhibitor arcs. In Helmut Jürgensen, Juhani Karhumäki, and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems - 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*, volume 8614 of *Lecture Notes in Computer Science*, pages 186–197. Springer, 2014. `doi:10.1007/978-3-319-09704-6_17`.

12 Jarkko Kari and Nicolas Ollinger. Periodicity and immortality in reversible computing. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2008. `doi:10.1007/978-3-540-85238-4_34`.

13 Ivan Korec. Small universal register machines. *Theor. Comput. Sci.*, 168(2):267–301, 1996. `doi:10.1016/S0304-3975(96)00080-1`.

14 Egor Kuzmin and Dmitry Chalyy. Decidability of boundedness problems for Minsky counter machines. *Autom. Control. Comput. Sci.*, 44(7):387–397, 2010. `doi:10.3103/S0146411610070047`.

15 Dominique Larchey-Wendling and Yannick Forster. Hilbert's tenth problem in Coq. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.FSCD.2019.27`.

16 Dominique Larchey-Wendling and Jean-François Monin. Simulating induction-recursion for partial algorithms. In *24th International Conference on Types for Proofs and Programs, TYPES 2018*, 2018.

17 Pierre Letouzey. A new Extraction for Coq. In Herman Geuvers and Freek Wiedijk, editors, *Types for Proofs and Programs, Second International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers*, volume 2646 of *Lecture Notes in Computer Science*, pages 200–219. Springer, 2002. `doi:10.1007/3-540-39185-1_12`.

18 Marvin Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics*, pages 437–455, 1961.

19 Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

20 Kenichi Morita. Universality of a reversible two-counter machine. *Theor. Comput. Sci.*, 168(2):303–320, 1996. `doi:10.1016/S0304-3975(96)00081-3`.

21 Kenichi Morita. Reversible computing systems, logic circuits, and cellular automata. In *Third International Conference on Networking and Computing, ICNC 2012, Okinawa, Japan, December 5-7, 2012*, pages 1–8. IEEE Computer Society, 2012. `doi:10.1109/ICNC.2012.10`.

22 Kenichi Morita. *Theory of Reversible Computing*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2017. `doi:10.1007/978-4-431-56606-9`.

23 Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, 1980. `doi:10.1145/322186.322198`.

24 Emanuele Rodaro and Pedro V. Silva. Amalgams of inverse semigroups and reversible two-counter machines. *Journal of Pure and Applied Algebra*, 217(4):585–597, 2013.

25 Matthieu Sozeau and Cyprien Mangin. Equations v1.2, May 2019. `doi:10.5281/zenodo.3012649`.

26 Simon Spies and Yannick Forster. Undecidability of higher-order unification formalised in Coq. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 143–157. ACM, 2020. doi:10.1145/3372885.3373832.

27 The Coq Development Team. The Coq proof assistant, January 2022. doi:10.5281/zenodo.5846982.